Proceedings of the ICTSS 2012 Ph.D. Workshop

Weise, Carsten; Nielsen, Brian

Publication date:
2012

Document Version
Accepted author manuscript, peer reviewed version

Proceedings of the ICTSS 2012 Ph.D. Workshop

Editors:
Carsten Weise and Brian Nielsen

Technical Report No. 12-201.
ISBN: 1601-0590.

Aalborg University
Department of Computer Science
Selma Lagerlöfsvej 300,
DK-9220 Aalborg
Denmark

Preface

This technical report contains the proceedings of the Ph.D. Workshop held in conjunction with the The 24th IFIP Int. Conference on Testing Software and Systems (ICTSS'12) in Aalborg, Denmark, November 19, 2012.

The well-established ICTSS series of international conferences addresses the conceptual, theoretic, and practical challenges of testing software systems, including communication protocols, services, distributed platforms, middleware, embedded systems, and security infrastructures.

The aims of the ICTSS Doctoral Workshop is to provide a forum for PhD students to present preliminary results and their thesis work and receive constructive feedback from experts in the field as well as from peers. Also it is an opportunity for researchers to get an insight into new research topics in the field. Ph.D. students at any stage of their doctoral studies may participate.

Seven abstracts were submitted. The submitted abstracts were reviewed and evaluated by 3 program committee members against the above goals. Four contributions were invited to be presented at the Workshop. It is the revised abstracts that are included in this report.

Aalborg, November 2012

Carsten Weise and Brian Nielsen (Editors)

Table of Contents

An Integrated Framework for Component-based Analysis of Architectural System Models

Raluca Marinescu, Cristina Seceleanu, and Paul Pettersson

Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University
Västerås, Sweden
{raluca.marinescu, cristina.seceleanu, paul.pettersson}@mdh.se

Abstract. Verifying architectural models of embedded systems is desirable, since architecture can impact the performance and resource usage of the final system implementation. To fulfill this need, one could think of combining formal verification and testing to achieve proofs of system correctness with respect to functional and extra-functional requirements. Our first step to accomplish this goal has concretized in the development of a framework that integrates architectural models described in East-adl language with component-based model-checking techniques. The framework is supported by a tool called ViTAL, which captures the behavior of East-adl functions as timed automata models, which can be formally verified in the Uppaal Port model-checker that exploits the components-based semantics at the architectural level. Later, the same formal models will help generate test-suites to provide support for model-based testing.

Keywords: East-adl, V&V techniques.

1 Introduction

Nowadays, many automotive functions are real-time, so a thorough Verification and Validation (V&V) is necessary to ensure real-time requirements at the architectural level. Current V&V tools are working isolated and their interaction is difficult [9], if not impossible. A smart combination of this different techniques, together with their successful application in industrial practice, could be the next step in the V&V evolution.

Lately, a lot of effort has been devoted to generate test-suites from system models (e.g., UML [4], Timed Automata (TA) [7]), and also to verify such models (e.g., Uppaal [1], PROGRESS [10]). However, these are sparse results with regard to the combination of V&V techniques.

The automotive industry provides its system specification in architectural description languages with no precise formal semantic, making it harder to use model-checking tools to analyze such embedded system (ES). In practice, some companies (e.g., VOLVO Technology AB and Continental Automotive) are using East-adl [2], an architecture description language dedicated to automotive ES,

which does not provide the possibility to construct, verify, and transform its models using formal techniques. Formal verification of both functional and timed behavior is necessary to ensure the real-time requirements at the architectural level, making EAST-ADL models a good basis for a combined V&V framework.

In our research, we intend to bring closer architectural description languages and verification techniques, through a new framework that consists of an integrated methodology that has been implemented in a tool called ViTAL [1](A **V**eri**f**ication **T**ool for EAST-**A**D**L** Models using UPPAAL PORT) [3], which provides Component-Based (CB) verification of EAST-ADL models via UPPAAL PORT. The tool lets one describe functional EAST-ADL behavior in TA semantics. To show the applicability of our tool and method, we illustrate its use on an industrial prototype, that is, Volvo Technology's Brake-by-Wire system. Later, ViTAL will be extended with test-suite generation capabilities to provide support for model-based testing, by generating test suites corresponding to various functional and extra-functional goals.

The paper is organized as follows. Section 2 briefly overviews EAST-ADL architectural language, UPPAAL PORT model-checker, and model-based testing. Section 3 presents the work already done and some preliminary results. In Section 4 we give a short description of the Brake-by-Wire industrial case study. Next, Section 5 describes our steps to finalize the proposed framework, before concluding the paper in Section 6.

2 Preliminaries

EAST-ADL. The architecture description language EAST-ADL is structured into five abstraction levels, which represent different stages of the engineering process, and provides traceability between them [2]. In addition, the structural organization of EAST-ADL has modeling constructs for behavior, requirements, timing, variability, and safety aspects. It captures structural components that refer to external or internal behavior as Simulink models.

UPPAAL PORT. Based on UPPAAL model-checker an extension for CB systems called UPPAAL PORT was released [5]. It uses local time semantics and a Partial Order Reduction Technique (PORT) to improve the efficiency of the verifier. UPPAAL PORT is suited for the analysis of EAST-ADL models because it assumes a "read-execute-write" component model semantics in its input language.

Model-based Testing (MBT). It derives test suites based on the specified functional requirements from a behavioral model of the system, covering one or more particular criteria [8]. A test harness executes the test suite against the implementation under test and the result is compared to the expected result, prescribed by the specification, by a test oracle. The test oracle delivers a verdict for each test case in the test suite.

[1] ViTAL is available at http://www.vitaltool.org

3 Contribution of the Thesis

The behavior of an EAST-ADL function prototype (FP) is described using external notations such as UML and Simulink, which do not have direct support for formally verifying EAST-ADL models. We propose a framework that integrates architectural description languages and verification techniques for CB ES, which have been implemented in the ViTAL tool. As depicted in Fig 1, the system designer creates the EAST-ADL models in a dedicated tool (e.g. Papyrus) and adds behavior to the EAST-ADL components, as TA models, such that UPPAAL PORT model-checker can be used to simulate the system model and verify various requirements (e.g., functional and timing requirements). We specify the internal behavior of each elementary FP as TA, and construct a complete system behavior model by the parallel composition of local behaviors. In addition, we map FP ports onto UPPAAL PORT read/write variables. A composition of function behaviors is considered a network TA that enables us to analyze and verify behaviors of the entire system using UPPAAL PORT model checker. To be able to perform this, we implement an automatic model transformation to UPPAAL PORT, which enables UPPAAL PORT to handle EAST-ADL models enriched with TA behavior as input.

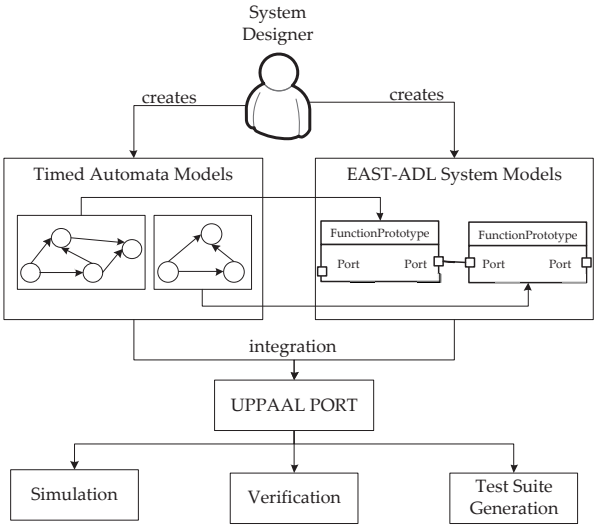


Fig. 1. The workflow of the integrated simulation and verification tool of EAST-ADL models

The above steps are implemented in our ViTAL tool, which provides an integrated environment for architectural description languages and verification techniques, based on different Eclipse plug-ins, as depicted in Fig. 2. The User

Interface integrates an editor for EAST-ADL models in the Eclipse framework, as well as a TA editor to model the timing and functional behavior of EAST-ADL FPs. UPPAAL PORT introduces support for simulation and verification, using a client-server architecture. The UPPAAL PORT model-checker consists of two modules: the Eclipse plug-in used as the graphical simulator, and the server executing the verification. Using the integrated simulator it is possible to validate the behavior and timing of an EAST-ADL system model, prior to design and implementation.

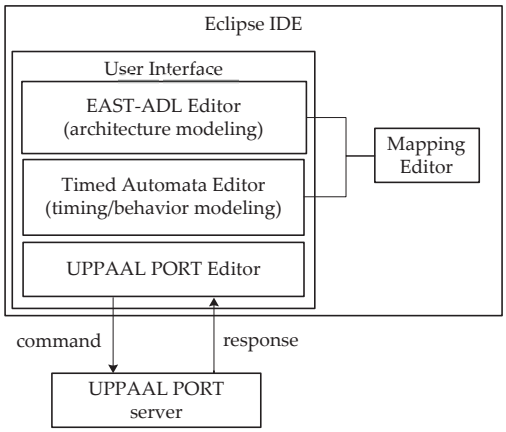


Fig. 2. Overview of the ViTAL tool architecture

In order to integrate the formal model of UPPAAL PORT TA with EAST-ADL, we need to first perform a semantic anchoring of the latter to a component model that obeys the *read-execute-write* semantics, hence preserving the informal semantics of EAST-ADL without altering its structure. The Mapping Editor shown in Fig. 2 can be seen as a function $\pi : EAST - ADL \rightarrow UPPAALPORT$, which maps each FP to an intermediate component, input ports to intermediate component data-flow input ports, output ports to the intermediate component data-flow output ports, connectors to the intermediate component connections, and the timing constraints to timing annotations.

ViTAL provides support only for the analysis of functional and timing requirements of EAST-ADL functions, but the limited software and hardware resources of complex automotive embedded systems require the analysis and verification of extra-functional requirements. Due to the lack of resource modeling notations in EAST-ADL, allocations of components cannot be analyzed and refined at earlier phases of design. To address this problem, we propose a modeling extension to the current EAST-ADL language with associated abstract resource information [6]. In order to annotate and reason about resource usage of EAST-ADL models, we need a semantic extension of the model and its behavior. At

the structural level, the resources are part of our extension of the EAST-ADL language in order to obtain resource awareness. At the behavioral level, Priced Timed Automata (PTA) can be used as a framework for the formal analysis of the corresponding models and the resource consumption represented as real-valued cost variables, and their evolution.

4 Applying ViTAL on the Brake-by-Wire System

The Brake-by-Wire (BBW) system consists of five Electronic Control Units (ECUs) connected by a network bus: a central ECU connected to the brake pedal and another four ECUs connected to each wheel. The central ECU has three components: a brake pedal sensor, a component that calculates the global brake torque from the brake pedal position, and a component that distributes the global torque to the four wheels. Each wheel ECU also has three components: a sensor that measures the wheel speed, a component for the brake actuator, and an ABS controller. The ABS controller is based on a simple logic: if the slip rate of the wheel is larger than 0.2, then the brake actuator is released and no brake is applied. Otherwise, the requested brake torque decided by the central ECU is used.

A set of properties concerning the safety and liveness of the BBW system have been verified with ViTAL. Here, we present a few representative properties that we have verified in our previous work [3]:

- The property of deadlock freedom;
- Timing properties, like the end-to-end deadlines;
- Functional properties, which relate to the slip rate value.

5 Future Work

To provide a real combination of V&V techniques, tailored for EAST-ADL architectural language, which is our main research goal, we plan to extend our framework with offline test suite generation capabilities for both functional and extra-functional testing goals. The tool support will be based on ViTAL and will use model-based testing to derive test-suites from EAST-ADL functions enriched with TA behavior models, by exploiting the trace information resulted as witnesses (or counter-examples) from UPPAAL PORT verification of appropriate properties.

To be able to carry out resource-wise analysis of EAST-ADL models, we intend to integrate our extension with a formal model, that supports resource analysis techniques for performing quantitative consumption analysis. We could show how analysis goals (e.g., feasibility analysis, optimal resource analysis) can be formalized and reasoned about by combining EAST-ADL with an abstract resource-aware behavioral model [6].

Last but not least, we intend to transform the abstract test-suites in executable test-suites and use them on the actual system implementation to be able to asses the effectiveness of our framework.

6 Conclusion

Our research goal of V&V of EAST-ADL models requires a consistent environment that brings together model-driven development, formal analysis, and test-suite generation. The employed formalism is the TA framework that captures the execution flow inside each FP and the complex interactions between components. In this paper, we have described a methodology towards the integration of EAST-ADL and UPPAAL PORT and its implementation in a tool called ViTAL. As future work, we will extend ViTAL with test-suite generation capabilities to enable the verification of EAST-ADL models. Through our framework, we hope to bring together the V&V techniques, tailored for architectural models of ES.

Acknowledgment: The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement number 269335 and from VINNOVA, the Swedish Governmental Agency for Innovation Systems, within the MBAT project.

References

1. Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, Lecture Notes in Computer Science. Springer, 1996.
2. MAENAD Consortium. East-adl domain model specification. http://www.maenad.eu/, 2011.
3. E.P. Enoiu, R. Marinescu, C. Seceleanu, and P. Pettersson. Vital: A verification tool for east-adl models using uppaal port. In *17th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2012.
4. S. Gnesi, D. Latella, and M. Massink. Formal test-case generation for uml statecharts. In *Ninth IEEE International Conference on Engineering Complex Computer Systems*, 2004.
5. John Håkansson, Jan Carlson, Aurelien Monot, and Paul Pettersson and. Component-based design and analysis of embedded systems with uppaal port. In *6th International Symposium on Automated Technology for Verification and Analysis*, 2008.
6. Raluca Marinescu and Eduard Paul Enoiu. Extending east-adl for modeling and analysis of system for resource-usage. In *Proceedings of the 4th IEEE International Workshop on Component-Based Design of Resource-Constrained Systems*. IEEE Computer Society Press, 2012.
7. Brian Nielsen and Arne Skou. Automated test generation from timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science. Springer, 2001.
8. A. Pretschner. Model-based testing. In *7th International Conference on Software Engineering (ICSE)*, 2005.
9. M. Utting and B. Legeard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2007.
10. A. Vulgarakis, J. Suryadevara, J. Carlson, C. Seceleanu, and P. Pettersson. Formal semantics of the procom real-time component model. In *35th Euromicro Conference on Software Engineering and Advanced Applications*, 2009.

Modeling and testing non deterministic real time systems

Ilham Kitouni[1], Djamel-Eddine Saidouni[1] and Kenza Bouaroudj[1]

[1] MISC Laboratory, University Mentouri
Constantine, 25000, Algeria
{ kitouni, saidouni, bouaroudj }@misc-umc.org

Abstract. The aim of this paper is to propose a new testing approach based on Timed Refusals Regions Graph (TRRG) in order to test non deterministic real time systems. Those systems are modeled by Durational Actions Timed Automata (DATA*). We characterize the DATA* model and we propose a framework to generate TRRG from DATA*. We discuss a technique to generate a canonical tester from TRRG. An implementation based on the combination of Meta-modeling and Graph Grammars, to transform a DATA* structure into a TRRG in the aim of creating a canonical tester and generating a test cases.

Keywords: Testing based models, refusals graphs, maximality semantics, non deterministic real time systems, canonical tester.

1 Introduction

The design and implementation of correct critical and real time systems is one of the major challenges of information technology. Formal testing can greatly increase the confidence in the functioning of these systems. It allows checking the correctness of a system with respect to its specification.

In this work we are interested in testing based models where the temporal behavior of systems is taken into account. Testing based on timed refusals allows the comparison between the behavior of the specification and the implementation, if the implementation refuses an action after a timed trace, the specification also refuses this action after the same trace. That means *I* and *S* have the same refusals sets and the same timed traces. This theoretical approach is necessary to generate testers.

Systems are modeled by durational action timed automata (DATA*). It is constructed on classical timed automata and augmented by maximality semantics. This later allows us i) *to carry durations of actions*, which is realistic assumption for specifying in a natural way systems and ii) *to handle true concurrency*. In [2] DATA* is proven to be a determinizable model and have suitable properties. From this point of view, it is well appropriate for modeling *real time concurrent and distributed systems*.

In the second time we propose a testing architecture based on the use of timed refusals regions graph structure (TRRG) for generating a canonical tester. The DATA* structure is determinized and decorated by refusals sets named Refusals DATA* (RDATA*), after the state space of RDATA* is reduced using an equivalence relation on regions, this step construct the TRRG [3].

In the next section, we define the DATA* model with properties. In section 3, test architecture is presented. Finally, we discuss some open issues in section 4.

2. DATA* Model

The DATA* model is a timed automata over an alphabet representing actions to be executed. This model takes into account, in the specification, the duration of actions based on an intuitive idea: temporal and structural non-atomicity of actions. This model seems interesting because it coated timed automata model by maximality semantics.

In the DATA* model, the actions durations are represented by constraints on the transitions and in the states. In this sense, any enabled transition represents the beginning of the action execution. On the target state of transition, a timed expression means that the action is possibly under execution (which is different from invariant in some class of timed automata); we recall that is a characteristic of maximality semantics based models.

Interleaved interpretations of concurrency are justified by assumption that all actions are atomic a direct consequence is that no two actions can occur simultaneously. In the opposite, maximality semantics based models present concurrent actions differently from choice [6]. As an illustration, consider the example depicted by Fig.1. In fact, information {x,y} on locations S2 and S4, in Fig.1.b make the difference and inform about the concurrent execution of actions (a and b).

From operational point of view, each clock is associated to an action. This clock is reset to 0 at the start of action and will be used in the construction of the temporal constraints as transitions guard.

- **Formalization**

Definition1 : A DATA* D is a tuple $(L, l_0, X, T_D, L_S, L_f)$ over ACT a finite set of actions, L is a finite set of locations, $l_0 \in L$ is the initial location, X is a finite set of variables named clocks and T_D is a set of edges. L_f is a subset of L for terminal locations. A clock takes values from R^+ or it is undefined, denoted by $\perp$. Without loss of generality, we write $R_\perp^+ = R^+ \cup \{\perp\}$ where the set of nonnegative real numbers is extended with the special value $\perp$. An edge $e = (l, G, a, x, l')$ represents a transition from location l to location l' on input symbol a, x is a clock to be reset with this transition. G is the corresponding guard. Finally, $L_S : L \to P(C_X)$ is a maximality function which decorates each location by a set of timed formula named: *Actions Durations*. Those concern overlapping execution of actions. C_X is a set of clock constraints over X.

Definition2: The semantic of a DATA* D is given by the timed transitions system (*TTS*): $(S_D, s_0, \to)$ over $ACT \cup R_\perp^+$. A state of S_D (or configuration) is a pair (l, v) such that l is a location and v is a valuation function over X, with initial configuration $(l_0, \perp)$. A terminal (accepting) configuration of *TTS* is a pair (l, v) with l in L_f.

Two types of transitions between configurations of S_D are possible and correspond respectively to time passing thus the run of transition from D.

In this work, we mainly focus on the decision problems and closure properties of DATA*. Hence, our aim is to characterize timed languages recognized by DATA* in terms of some suitable deterministic class of timed automata. We also show that DATA* are closed under Boolean operations. We investigate the expressive power of DATA*, we show that the DATA*$_0$, which are a DATA* with null durations, are expressively equivalent to Event Recording Automata [5]. However, the known strict inclusion of DATA*$_0$ in DATA* seems to result into a new map for inclusion in the class of timed languages.

After we give a technique of reducing regions automaton of DATA* in an aggregated regions automaton to avoid explosion state space. We show that there is a homomorphism on the behaviors of the two automata [1].

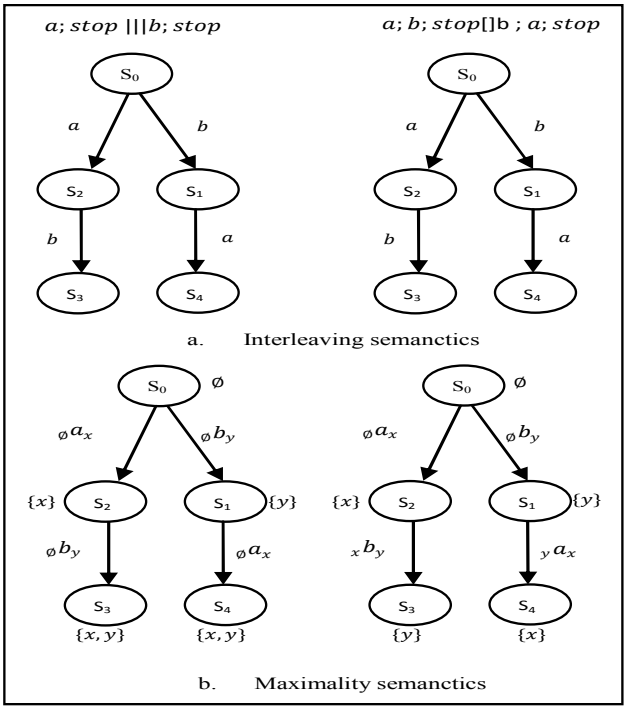


Fig 1: Representation of concurrency and choice in maximality semantics and interleaving semantics.

Illustrate the DATA* model with the example above (Fig.2):

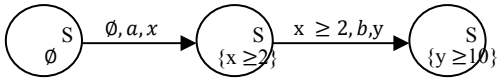


Fig 2: DATA*.

3 Test Architecture

We propose a new testing architecture for testing systems modeled by DATA* the aim is to construct a canonical tester by several transformations. Moreover, we investigated the automatic extraction of test cases. A detailed presentation of this approach is in [3]. We summarize it as follow:

1- Construction of RDATA* from specification automaton. 2-Construction and reduction of regions graph from RDATA* named TRRG, based on aggregation method established for timed automata in [1]. 3- Generation of canonical tester from TRRG graph. 4- Automatic extraction of test cases based on coverage criteria.

- **Refusals DATA*:** is a deterministic DATA* extended by refusals sets:

Definition3: $\mathrm{RD} = (\mathrm{D}, \mathrm{Ref_{TPR}})$ with $\mathrm{D}= (L, l_0, X, T_D, L_S, L_f)$ is a deterministic DATA* over *Act* and $\mathrm{Ref_{TPR}}: \mathrm{L} \rightarrow \mathrm{P}\left(\mathrm{P}\left(\overline{\mathrm{Act}} \cup \overline{\overline{\mathrm{Act}}}\right)\right)$ is an application that associates for any $\mathrm{l} \in \mathrm{L}$ a set of refusals. $\overline{\mathrm{Act}} = \{(\bar{\mathrm{a}}, \mathrm{G}): \mathrm{a} \in \mathrm{Act}\}$ and $\overline{\overline{\mathrm{Act}}} = \{(\bar{\bar{\mathrm{a}}}, \mathrm{G}): \mathrm{a} \in \mathrm{ACT}\}$.

We define the timed permanent and timed temporary refusals sets : $\overline{\mathrm{V}} = \{(\bar{\mathrm{a}}, \mathrm{G})\} \in \mathrm{Ref_{TPR}(l)}$ as timed permanent refusal. It means that the action *a* may be refused permanently from the state *l*, this refusal is possible but not certain. This certitude will take place after the satisfaction of guard G. this refusal results from determinization operation of DATA* structure. $\overline{\overline{\mathrm{V}}} = \{(\bar{\bar{\mathrm{a}}}, \mathrm{G})\} \in \mathrm{Ref_{TPR}(l)}$: as timed temporary refusal and it means that action is refused as much as the guard G is not satisfied. This refusal results from durational actions hypothesis. $\mathrm{P}\left(\mathrm{P}\left(\overline{\mathrm{V}} \cup \overline{\overline{\mathrm{V}}}\right)\right)$ is partition of parts of refusals sets. Based on determinization property of DATA* model and definition of different kind of refusals the construction of RDATA* is done.

- **Minimization of Refusals Sets:** The minimization procedure of refusals sets eliminates redundant information about refusals at any location of RDATA*.

Definition4: Let $RD = (D, Ref_{TPR})$ *be a RDATA*,* $l \in L$ *and let* $A, B \in Ref_{TPR}(l)$, $A \Subset B$: $\forall(\bar{a}, \emptyset) \in A$, $(\bar{a}, \emptyset) \in B$ *and* $\forall(\bar{\bar{a}}, G) \in A$ *either* $(\bar{\bar{a}}, G) \in B$ *or* $(\bar{a}, \emptyset) \in B$.

The minimization of refusals set *A* produces a new set *A'* calculated for any location $l \in L$ is as follows:

1- $\forall A \in Ref_{TPR}(l) if\ (\bar{a}, \emptyset) \in A$ *and* $(\bar{\bar{a}}, G) \in A$ *then remove* $(\bar{\bar{a}}, G) from\ A$
2- Minimize $Ref_{TPR}(l)$ with respect to the relation $\Subset$.

- **Timed Refusals Regions Graph:** In previous work [1], we have defined an aggregation operation on regions automaton states for timed automata based on observable traces, using an equivalence relation. This aggregation reduces significantly the graph size. For this purpose we have adapted the proposed algorithm to generate aggregated regions automaton for DATA* which preserves the reachability property.

While the regions graph associated to the DATA* was creating, symmetrical aspects of clock regions is revealed. Indeed, because of the causal dependence of actions when considering durations of actions, the guards of the transitions have a particular form also the single clock reset, in the beginning of action execution. These two characteristics allow us deducting the form of regions and their successors which verify guards and clock rest at each point of time.

This detailed principle is used to construct and reduce in the same time de timed refusals regions graph TRRG.

Testing with TRRG [3]: In our case we introduce the use of TRRG in testing timed systems specified by DATA*. Therefore, the conformance relation must be decided and we have to take into account actions which elapse in addition to temporal requirements. The *TRRG* structure permits to define a timed extension of conformance relation based on classical *conf* relation for DATA* defined in [9]. This relation was widely used in the practice of the test on Labeled Transitions Systems.

We define a timed conformance relation named $conf_{tpr}$ as follows:

Definition5: σ is a timed trace , $I\ conf_{TPR}\ S \stackrel{\text{def}}{=} \forall \sigma \in Ttraces(S)$ $\left(Forb(I,\sigma) \subseteq Forb(S,\sigma)\right) and\ \left(Ref_{TPR}(I,\sigma) \subseteq Ref_{TPR}(S,\sigma)\right)$.

The use of this notion of conformance makes DATA* more expressive. And $conf_{TPR}$ can be refined and used explicitly for creating a tester for deriving test cases.

. **Implementation [4]:** The proposed approach was implemented using graph transformation. Which is a process converts a model to another model. This task requires a set of rules (Graph Grammars) that define how the source model has to be analyzed and transformed into other elements of the target model. For this purpose AToM^3 is used.

5 Conclusion and future Work

The consideration of temporary refusals in testing is a question that has been addressed in the literature since 1981 [12]. For instance, Langerak [11] considers system which may refuses some actions, however these refusals may disappear after applying extra events on it. In this theory, the origin of temporary refusals is unknown and extra events are needed to eliminate this lock.

Tretmans in [10] has defined the notion of quiescence in system behaviors, this situation may occur when a system executes a cyclic sequence of silent actions. To distinguish between quiescence situation and temporary refusals, Brinksma and al propose in [13] an extension of the conformance relation for real time systems and introduce a notion of quiescence parameterized by upper bound of duration for this lock. While this period has not expired, the refusal may be temporary, the system is considered in a quiescence location.

Nielsen proposes in [8], an approach for testing timed systems based on a Hennessy's testing theory [7] specifications are defined as event recording automata over a given finite set of actions. The specification structure is converted to a trace equivalent *deterministic* state machine whose states are labeled with the must sets for that state. A simple timed generalization of Hennessy's tests. This theory is based on tree abstraction defined as: after σ must A, after σ must Ø and canσ; σ become a timed trace (a sequence of alternating actions and time delays), after which an action in A must be accepted immediately for example.

Our proposition presents similarity from Nielsen one, the difference consists in the conformance relation and the model used for specification. We think that our approach is more general and rich in the sense that it combines, advantages from maximality semantics, non deterministic timed models and refusals testing.

A lot of works remain to be done; we plan to use this result in order to construct a validating approach for R.T. systems, and to define how to select complete tests and the possibility to combine model checking algorithms and refusals testing.

References

1. I. Kitouni, H. Hachichi, D.E. Saidouni: A Simple Approach for Reducing Timed Automata. In: The 2nd IEEE International Conference on Information Technology and e-Services (ICITeS 2012). Sousse, Tunisia (March 24-26, 2012).
2. I. Kitouni, H. Hachichi, K. Bouaroudj and D.E. Saidouni: Durational Actions Timed Automata: Determinization and Expressiveness. In: International Journal of Applied Information Systems (IJAIS) 4(2):1-11, September 2012. Published by foundation of Computer Science. New York, USA.
3. I. Kitouni, H. Hachichi, K. Bouaroudj and D.E. Saidouni: Timed Refusals Graph for Non-Deterministic Timed Systems. In: International Journal of Computer Science and Telecommunications (IJCST). Volume 3, Issue 9, September 2012.
4. H. Hachichi, I. Kitouni, K. Bouaroudj and D.E. Saidouni: A Graph Transformation Approach for Testing Timed Systems. In: The 18th International Conference on Information and Software Technologies (ICIST 2012) published as a volume of Springer-Verlag CCIS 319, pp. 123–137, Kaunas, Lithuania (September 13th - 14th, 2012).
5. R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: a determinizable class of timed automata. Theoretical Computer Science, 211(1–2):253–273, (1999)
6. H. Bowman and R. Gomez: Concurrency Theory, Calculi and Automata for Modelling Untimed and Timed Concurrent Systems. ISBN-10: 1-85233-895-4 ISBN-13: 978-1-85233-895-4 Springer-Verlag London Limited (2006)
7. R. D. Nicola and M. Hennessy. Testing Equivalences for Processes. Theoretical Computer Science, 34:83–133, (1984)
8. B. Nielsen. Specification and Test of Real-Time Systems. PhD thesis, Department of computer Science, Aalborg University, Denmark, april 2000.
9. E. Brinksma , theory for the derivation of tests. In S. Aggarwal and K. Sabnani, editors, Proceedings of the 8th IFIP, Symposium on Protocol Specification, Testing and Verification (PSTV 1988). North-Holland, (1989).
10. J. Tretmans, Test generation with inputs,outputs and repetitive quiescence. In Software-Concepts and Tools,17(3) (1996)
11. R. Langerak, Transformations and Semantics for LOTOS. PhD thesis, University of Twente, Netherland, (1992)
12. I. Phillips, refusal testing ; theoretical computer science : 241-284,1987. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195-197 (1981)
13. E. Brinksma, L. Brandan, and Briones. Test generation framework for quiescent real time systems. In J. Grabowski and B. Nielsen, editors, FATES, volume 3395 of LNCS, pages 64-78, Berlin Heidelberg, Springer-Verlag, (2005)

Filtered Comparison for Oracle in Model Transformation Testing

Olivier Finot (PhD Student),
Jean-Marie Mottu, Gerson Sunyé, and Christian Attiogbe (Advisors)

LINA CNRS UMR 6241
University of Nantes
2, rue de la Houssinière
F-44322 Nantes Cedex, France
olivier.finot@univ-nantes.fr

Abstract. Focusing on one part of a produced output helps in improving model transformation testing

1 Introduction

Models are becoming a key element in software engineering. With Model Driven Engineering (MDE), they are the heart of the development process. They are used to describe a system at some state of its development, and they evolve with transformations. Model transformations can be chained, up to the production of executable code.
However, any error in a transformation of such a chain will be spread to the resulting code. But while testing the produced code might detect the bug, finding its origin will be difficult. Therefore, it would be useful to check the chain's development by verifying the transformations.
The subject of this PhD thesis is the study of test oracles in MDE, and particularly for model transformation testing. Several contributions have already been published on the verification of model transformation. Our goal is to pursue research on this field and improve existing methods to test model transformations.
In Section 2, we present part of our work on the subject of model transformation testing. We propose a new approach to build a partial test oracle.[1]. Then, in Section 3 we discuss current and future work. Finally we conclude in Section 4.

2 Partial Oracle for Model Transformation Testing

Model transformations are automated to be highly reused. If we want to reuse a piece of software, we have to trust it. We can not have any errors in something we will reuse numerous times. We use test to ensure the correctness of a model transformation w.r.t. its specification.
The tester provides a valid input model, then she executes the Transformation Under Test (TUT) over this input model. Finally the test oracle

[1] a paper on the subject is currently under review for ICST13

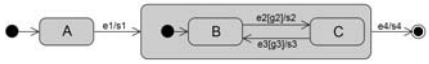
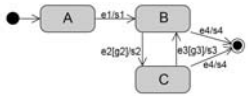
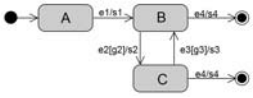


Fig. 1. Example M^{in}, of Hierarchical State Machine



(a) Variant M_1^{out} with only One Final State



(b) Variant M_2^{out} with Two Final States

Fig. 2. Possible Results for the Flattening of M^{in}

is what we are interested in, it controls the output model produced by the TUT's execution.

We are particularly interested in oracles for model transformation testing. While several studies discuss the generation and selection of input models [1] [2] [3] [4], the oracle is seldom considered [5] [6].

In some cases, the transformation's output is particularly complex. Thus, building an oracle controlling such an output is all the more difficult. An example of such a complex output is the case of polymorphic outputs. In this case the transformation's specification allows several valid variants of a given output. For example if we consider a program running an operation on a Finite State Machine (FSM) in order to flatten it, the input of this program is transformed into another FSM expressing the same behavior without using any composite state. We can transform the input model presented in Figure 1 into the output model depicted in Figure 2(a). With such state machines, the number of final states is not limited to only one. Thus, the FSM presented in Figure 2(b) is also a correct output for the flattening of the FSM presented in Figure 1.

While the implementation of such a transformation is deterministic, the specification allows several variants. The developer implements only one of these variants. However, when building an oracle, the tester must not consider the transformation's implementation, since she might be influenced by errors made by the developer. Thus, she has to design an oracle that checks that the produced output of the Transformation Under Test is one of the possible variants.

Classically, building a test oracle for model transformation consists in comparing the produced output model with a reference one [5]. The reference output model is the output model expected for the correct transformation of the input. Applying this approach to model transformations with polymorphic outputs would mean having the tester define one reference output model for each of the variants and then compare the produced output model to all of them. If the produced output model is identical to one of the reference output models, the test passes; otherwise it fails. Applying this classical approach to polymorphic outputs is

time-and effort-consuming for the tester. She has to manually define all the reference outputs, and run the comparisons.
In [7], we propose a more efficient approach to build an oracle to test model transformations with polymorphic outputs. We notice that in the polymorphic outputs of a model transformation some elements do not change from one variant to the other. In the example of the Figure 2, this is the case for the initial and simple states as well as for the transitions between them, they belong to the common part of the variants; the other elements form the variable part.
Our idea is to build an oracle focussing on this common part. Since by definition, the common part is identical in all the variants, this oracle will only need one reference output model as oracle data. The produced output model is compared to the reference one. The result of this comparison is then filtered in order to eliminate any difference concerning the variable part of the output model. If the filtered result of the comparison is empty, the common parts of both the produced and reference output model are identical, the test passes. Otherwise, the produced output model contains errors, an then the test fails.
Figure 3 summarizes our approach. We provide as oracle data the reference output model as well as patterns. In order to eliminate the differences about the variable part, we need to know which elements belong to this variable part. We define these elements according to their types (e.g. their meta-classes). The patterns we provide are meta-model excerpts defining the variable part of the transformation's output.
Our approach, requires the tester to identify the common and variable parts of the transformation's polymorphic outputs. A transformation has polymorphic outputs because the specification allows several syntaxes for a given semantics. Either it is clearly stated which elements are the source of this polymorphism, or this piece of information can be found in the output meta-model.
Languages, for instance, usually contain binary operators such as the logical or for which the order of the operands does not matter. Therefore a model instance of this language can have several variants by modifying this order; here the variable part is composed of all the instances of the operator and their operands. For instance, Bisztray et al. [8] transform UML activity diagrams into modelized CSP programs; the BinaryOperators and their instances form the variable part. The common part is identified only once for all test cases of a given transformation.

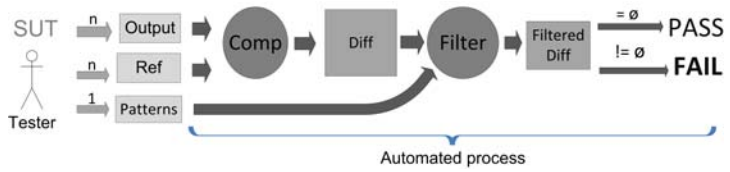


Fig. 3. Our Approach to Build a Partial Oracle

In our approach, we focus on controlling one part the produced output, producing a partial verdict. Nevertheless, a partial verdict is already a good piece of information. We are able to detect errors in the produced

output model using only one reference output model, whereas the classical approach requires to define as many reference output models as correct variants of the output model.
This approach has been automatized and we ran experiments on two different case studies. We concluded that our approach requires the tester to define less model elements, than with the classical approach. In those case studies, the common part is more important than the variable one This work has been submitted to ICST 2013 and is currently under review.

3 Ongoing Work

This PhD's subject is test oracles in MDE. We discussed in Section 2 one contribution aimed at partially controlling the polymorphic outputs of a model transformation.
Another part of the work on this subject consists in completing the obtained partial verdict, by controlling the unchecked variable part of the produced outputs. Whereas for the common part we use a reference output model, the idea here is to work directly on the produced output model. The tester starts by checking that the expected elements for this variable part are present in the model (in Figure 2, transitions from the states B and C towards final states). Then she ensures that there is nothing else in the variable part (no other instance of the meta-model fragments used for the filter).
Model transformations do not always produce polymorphic outputs. Also, defining a comprehensive reference output model can still be difficult even when it is not polymorphic. For instance, the larger the handled models become, the harder it is for the tester to define a reference output model. It is time-and effort-consuming for her to manually define a large and often complicated model. However it is easier to produce a partial reference output model focussing on some elements. With our approach, she could use this partial reference output model to obtain a partial verdict. This partial verdict can be a useful piece of information.
Outside the scope of model transformations the tester can still be faced with the complexity of the produced outputs. It can be difficult to manually produce a comprehensive reference output when dealing with large graphs or databases. Complex outputs are not just big sets with many properties, these properties are organised and structured. Applying our approach the tester can define partial oracles controlling such complex outputs. She only needs a partial reference output and a definition of the part she would not be interested in. In the case of regression testing, the partial reference output is produced by the previous version of the System Under Test.

4 Conclusion

The topic of this PhD thesis is about test in a model driven engineering environment. Our first contribution is the proposal of an efficient

approach to build a partial oracle controlling the common part of polymorphic outputs in a model driven environment. We are currently working on controlling the remaining, variable part. Also while the proposed approach was defined to control the polymorphic outputs of a model transformation, we are currently studying its to other model transformations. Afterwards we will confront our approach to other programs with complex output data.

References

1. Sen, S., Baudry, B., Mottu, J.M.: On combining multi-formalism knowledge to select models for model transformation testing. In: ICST. (2008) 328–337
2. Fleurey, F., Baudry, B., Muller, P.A., Le Traon, Y.: Qualifying input test data for model transformations. SOSYM (2009)
3. Mottu, J.M., Sen, S., Tisi, M., Cabot, J.: Static analysis of model transformations for effective test generation. In: ISSRE. (2012)
4. González, C.A., Cabot, J.: Atltest: A white-box test generation approach for ATL transformations. In: MoDELS. (2012) 449–464
5. Mottu, J.M., Baudry, B., Le Traon, Y.: Model transformation testing : oracle issue. In: MoDeVVa. (2008)
6. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. ECEASST (2009)
7. Finot, O., Mottu, J.M., Sunye, G., Attiogbe, C.: Comparaison de modèles filtrée pour le test de transformations de modèles. In: Conférence en IngénieriE du Logiciel CIEL. (2012)
8. Bisztray, D., Ehrig, K., Heckel, R.: Case study: Uml to csp transformation. AGTIVE (2007)

Relating Variability Modeling and Model-Based Testing for Software Product Lines Testing

Hamza Samih

hamza.samih@inria.fr

Abstract. More and more industries must build and simultaneously maintain several variants of a system in order to satisfy different user requirements. Software Product Line (SPL) engineering aims at identifying and managing commonality and variability (i.e., differences) among a set of variants. The development of an SPL needs solutions for including the testing activity. In this work, we explore how Model-Based Testing (MBT) can be applied in the context of SPL engineering. MBT is effective for testing a single system. Currently this technique cannot handle variability in test generation. To raise this limitation, we propose an approach based on functional requirements to link a variability model with a test model realized with MaTeLo MBT tool. The aim is to generate a test model for a selected member product of SPL.

1 Introduction

The extreme diversity of users and requirements for software systems forces software industry to increase the degree of variability and adaptability of their products. A growing number of companies adopt a SPL approach to deal with this major change in software development. This approach usually consists in designing a variability model, which captures all common and variable parts of the SPL. Then, software architects can decide on which variants to choose in order to derive a specific product in the line. SPL is a promising approach to increase reuse of core software assets, systematically document variability and eventually improve time-to-market for variants of a given system. However, from a testing point of view, SPL represents a major challenge. In particular, model-based testing does not support the notion of variability to automatically generate test cases for specific products [3] [4].

This PhD project addresses one core challenge: how can we reconcile variability modelling and model-based testing in order to reuse test models for the automatic generation of product-specific test cases. Recent work has addressed a part of this question by adapting combinatorial test selection to variability models. Hervieu et al. [5] and Perrouin et al. [7] propose different techniques that select a subset of all possible products for testing, based on pairwise coverage of interactions between variants in the product line. However, these techniques do not generate the test cases for the selected products.

In this work, we rely on OVM (Orthogonal Variability Model) [3], [4] for variability modelling and MaTeLo [1]for model based testing.

The MaTeLo Model-Based Testing is a tool developed by ALL4TEC French SME, its approach is to optimize the test process and improve the systems validation thanks to Markov chains based usage models to generate automatically test cases [1][2]. ALL4TEC look to introduce the variability in its test approach based on Model-Based testing by creating one usage model to test a product line. To validate that assumption we are faced to two challenges.

1. The first challenge is to link OVM model with MaTeLo test model of product line by managing the traceability between the variability and the equivalent elements of test model. This traceability is based on functional requirements of software system, which each requirement must be linked to equivalent features and to equivalent elements of test model.
2. The second challenge is to extract automatically an equivalent test model to a selected product of SPL. The core challenge is to extract a model that contains all equivalent test elements that describe the behavioral of the selected product features, as well as to be complete and valid model on the point of view Markov chain.

In the next section, we present the formalisms and the SPL testing approach we will use in the PhD work.

[1] The acronym for "Markov Test Logic". An MBT tool that is dedicated for building usage models and generating test cases. It is developed by ALL4TEC

2 SPL Testing by MBT

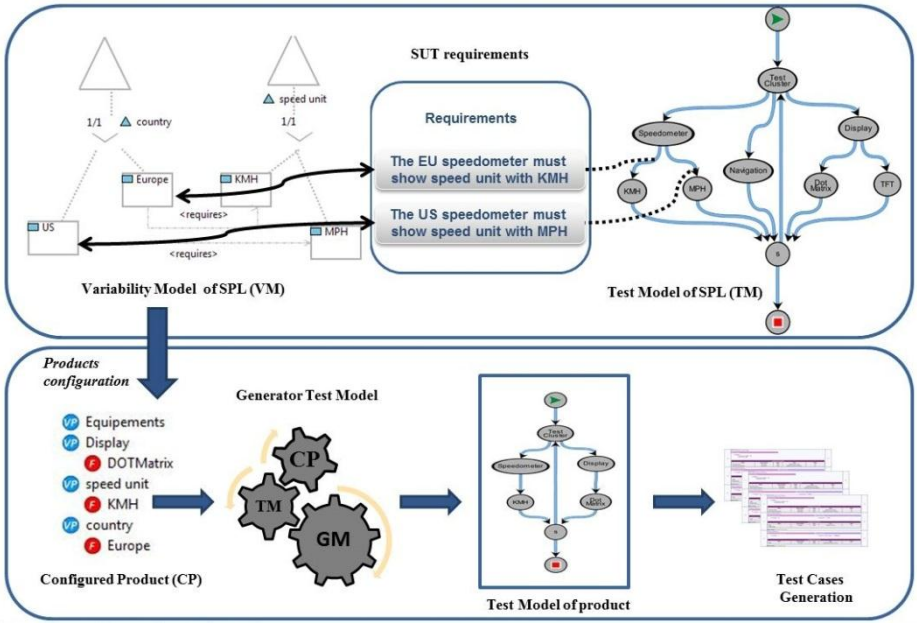


Figure. 1. SPL Testing by MBT

The first phase proceeds in three steps. The first step in that approach is realizing the test model of SPL. On MaTeLo we can associate functional requirements described in naturel language on each transition of the test model, to check in the test generation whether is covered it or not (right Top on the **Fig**.1) [1]. The second step is to model the variability of product line (left Top on the **Fig**.1) using OVM figures.

The OVM is a formalism to document only the variability of SPL, in this model the all way may vary a product line are figured by variation point (VP) and all instance of VPs are figured by variant (V) . The Vs are no more than features, which help to configure the valid products while enforcing the constraints between VPs, V or both [4]. The top left Figure 1 depicts the OVM model.

The last step is linking the both formalisms, the MaTeLo test model with OVM using the requirements documentation. The traceability is assured by linking manually each requirement that described variability of SPL, to equivalent feature on OVM and link it also to the equivalent transition on the MaTeLo test model (Top of **Fig**.1).

The second phase is generating a test model for selected valid product, but before we need first to configure manually all products from the OVM model (Left bottom of **Fig**.1). The products configuration is based on selecting all desired features to compose the product under development in accordance with constraints between VPs and Vs. The configured products are used as input to algorithm of test model generator. We select the product to generate its test model. The general principle of the algorithm is as follows:

1. Extract all requirements that satisfy all composed features of selected product.
2. Select all tagged transitions within MaTeLo test model by the selected requirements in the first iteration.
3. Delete all not matched elements of test model to selected product.
4. Update the rest of test model for selected product to be a valid Markov chain model.

Finally with MaTeLo tool, we can derive for the generated test model of selected product the equivalent test cases (bottom part of **Fig**.1).

3 Conclusion and perspective

In this work we propose a new technic to test SPL with MBT, by enriching the test model with the variability thanks to traceability with functional requirements tagged in the both formalisms.

Currently the association requirements – features, requirements – transitions and product configuration is done manually. Therefore we need to reason about consistency and for implementing an efficient solution to derive automatically and safely a "test model".

We plan to do an industrial experimentation in European project MBAT[2] where we are involved, to validate that assumption the realization a test model for SPL and the proposal of extracting a test model for a valid product of realistic SPL.

REFERENCES

1. Le Guen, H. (2003, septembre). Thelin. Practical experiences about statistical usage testing. In STEP. Amsterdam.
2. Mark Utting, B. L. (2007). *Pratical Model-Based testing A tools Approach.* ISBN-13: 978-0123725011
3. Disambiguating the Documentation of Variability in Software Product Lines:A Separation of Concerns, Formalization and Automated Analysis. Univ of Duisburg Essen,Essen Heymans,P. ; Pohl,K. ; Schobbens,P.Y. ; Saval,G. **Page(s):** 243 - 253
4. Pohl, K., Bockle, G., and van der Linden, F., Software Product Line Engineering : Foundations,Principles, and Techniques, Springer (2005)
5. A. Hervieu, B. Baudry, and A. Gotlieb, "Pacogen: Automatic generation of pairwise test configurations from feature models," in Proc. of the 22nd IEEE Int. Symp. on Softw. Reliability Engineering (ISSRE'11), Hiroshima, Japan, Nov. 2011
6. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[2] The Combined Model-based Analysis and Testing of Embedded Systems is one of ARTEMIS European projects

7. Perrouin, G., Sen, S., Klein, J., Baudry, B., Traon, Y.L.: Automated and scalable t-wise test case generation strategies for software product lines. In: Third International Conference on Software Testing, Verification and Validation (2010)